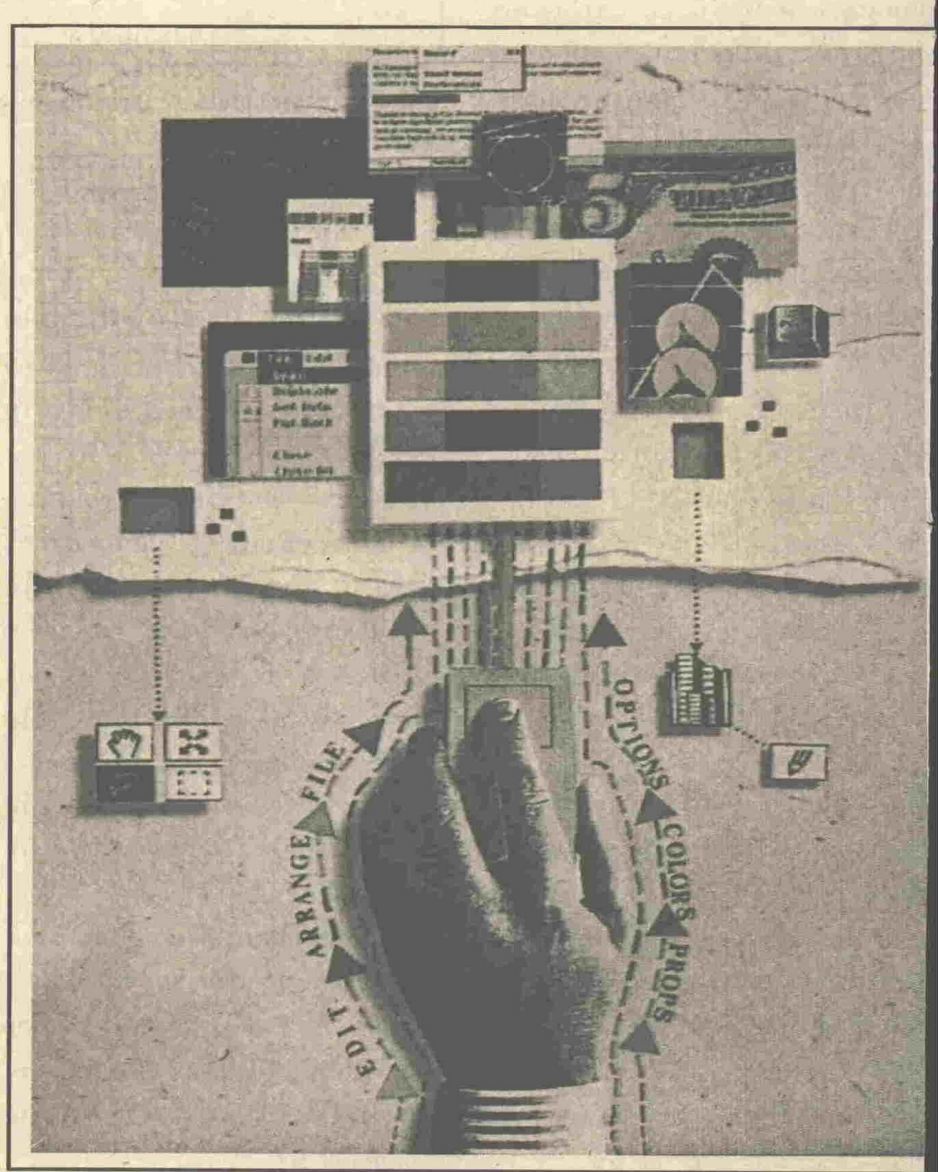
ร่ายยาว GUIs
สิ่งที่ควรรู้เกี่ยวกับวินโดว์ (2)

ยิ่งขุดยิ่งลึก
แค่นึกก็เห็นภาพ
การทำงานใน GUIs



ในตอนที่แล้วผมได้กล่าวถึง GDI (Graphics Device Interface) ไปบางส่วน ซึ่งพอจะสรุปหน้าที่หลักได้ 3 ประการคือ

1) GDI มีหน้าที่เตรียมการเชื่อมต่อระหว่างแอพพลิเคชันกับอุปกรณ์ ซึ่งตัวเชื่อมรอยต่อนี้ เรียกว่าดิสเพลย์คอนเท็กซ์ (Display Context)

2) GDI จะจัดเตรียมชุดคำสั่งสำหรับการวาด (drawing routine) เพื่อแสดงข้อความหรือรูปบน Client Area

3) GDI จะจัดเตรียมชุดของรูทีนที่ใช้ในการสร้าง tools ต่าง ๆ ซึ่งจะเรียกใช้โดยดรอว์อิงรูทีนอีกทีเพื่อทำการวาดภาพต่อไป

จากที่เราได้กล่าวไว้ในฉบับที่แล้วว่า GDI output functions จะต้องใช้ tools ต่าง ๆ คือ ปากกา (pen), แปรง (brush) และตัวอักษร (font) ในการทำงาน แต่มีสิ่งหนึ่งที่ต้องระลึกไว้เสมอคือ tools เหล่านี้ถือเป็นออปเจ็กต์ซึ่งไม่สามารถแก้ไขได้ (nonmodifiable objects) หากแต่ใช้วิธีการเลือกเปลี่ยนเมื่อต้องการใช้อุปกรณ์ตัวอื่น เช่น ต้องการเปลี่ยนจากปากกาสีแดงไปใช้ปากกาสีเหลืองก็ต้องทำการสร้างปากกา (CreatePen) สีเหลืองขึ้นมาก่อน แล้วถึงจะเปลี่ยนไปใช้แทนปากกาสีแดงได้ โดยการ SelectObject และเมื่อใช้เสร็จก็ทำลายเสียด้วย DeleteObject (เสียดายปากกาจัง) สาเหตุที่ tools เหล่านี้ไม่สามารถแก้ไขได้ก็เพราะมันถูกจัดเป็น stock object ที่ใช้ในระบบซึ่งไม่สมควรที่จะยอมให้แก้ไขได้ (และความจริง CreatePen ไม่ได้สร้างปากกาขึ้นมาใหม่จริง ๆ หรอกครับ หากแต่เป็นการสร้าง handle

องปากกาที่จะส่งให้แอพพลิเคชันไป
ช้ต่างหาก) ในตารางที่ 1 เป็นการแสดง
ารวาดรูปหลายเหลี่ยม โดยใช้ GDI
rawing routine (จัดอยู่ในเอาต์พุต-
งก์ชัน) ที่ชื่อ Rectangle ซึ่งในรูทีน
าดอันนี้ ต้องใช้ปากกาในการวาดเส้น
อบและใช้แปรงในการวาดและระบาย
ภายใน จากรูปเป็นการแสดงทางเลือก
ากกาและแปรงแบบต่าง ๆ

PEN STYLE	BRUSH STYLE	OUT PUT
PS-SOLID	SOLID (สีดำ)	
PS-DASH	SOLID (สีขาว)	
PS-DOT	SOLID (สีขาว)	
PS-DASHDOT	SOLID (สีขาว)	
PS-DASHDOTDOT	SOLID (สีขาว)	
PS-SOLID	HS-BDIAGONAL	

ตารางที่ 1 แสดงตัวอย่างเอาต์พุตของฟังก์ชัน Rectangle เมื่อเลือกใช้ Pen และ Bruch ต่างกัน

าด้วยการแสดงผล

ตลอดเวลาที่ผ่านมา เราพูดถึง
C ในเชิงความหมายของ Display
ontext ซึ่งความจริงแล้ว DC นี้อาจมี
วามหมายไปถึง Device Context ด้วย
ได้โดยดิสเพลย์คอนเท็กซ์เป็นดีไวซ์-
อนเท็กซ์แบบพิเศษชนิดหนึ่ง สำหรับ
ารแสดงผลบนไคลเอนต์แอเรียของ
นโดว์หนึ่ง ๆ ในขณะที่ดีไวซ์คอนเท็กซ์
ะระบุถึงอุปกรณ์การวาด ข้อมูลการ
าดและอื่น ๆ สำหรับอุปกรณ์จริง เช่น
อภาพ เครื่องพิมพ์ แต่ในดิสเพลย์-
อนเท็กซ์จะระบุเฉพาะสิ่งที่จำเป็น
าหรับไคลเอนต์แอเรียเท่านั้น

ือพึงระลึกเกี่ยวกับการ
สดงผลของ GDI

ก่อนที่จะขึ้นหัวข้อเกี่ยวกับเทคนิค
เศษที่มีความสัมพันธ์อย่างใกล้ชิดคือ
tmaps, clipping และ mapping จะ
อสรุปสิ่งที่สำคัญในการใช้ GDI out-
ut functions ดังนี้

- GDI output functions ต้องการ
ndle ของดิสเพลย์คอนเท็กซ์เสมอ
ยนิยมตั้งชื่อตัวแปรว่า hDC (handle
isplay Context)

- DC เป็นโครงสร้างข้อมูลแบบหนึ่งมีหน้าที่เก็บข้อมูลที่จำเป็นต้องใช้ให้กับ GDI output functions โดยจะถูกเซตค่า default เมื่อถูกสร้างขึ้นมาโดยใช้ฟังก์ชัน CreateDC (ซึ่งจะคืนค่า handle ของ DC กลับมาให้)

- DC จะมีข้อมูลเก็บไว้หลาย ๆ อย่างเช่น

1) a font handle
2) a pen handle
3) a brush handle
4) a map mode

- GDI functions จะใช้ลอจิกคัลโคออดิเนตซึ่งจะถูกแม็พไปสู่จอภาพอีกครั้งเมื่อทำการแสดงผล

กลวิธีสร้างภาพ (BITMAP)

โดยทั่ว ๆ ไปคำว่า บิตแม็พ จะหมายถึง ภาพ (image) ที่ถูกสร้างขึ้นจากเพตเทอร์นแบบต่าง ๆ ของบิตจำนวนมหาศาล ซึ่งสามารถแบ่งกว้าง ๆ ได้เป็น 2 ชนิด

1) device-dependent bitmap เป็นบิตแม็พที่มีความสัมพันธ์ใกล้ชิดระหว่างการจัดเรียงบิตบนหน่วยความจำกับอุปกรณ์ ตัวอย่างที่พบทั่วไปคือบิตแม็พบนจอภาพ

2) device-independent bitmap เป็นบิตแม็พที่มุ่งไปที่การสร้างภาพมากกว่ารูปแบบการจัดเรียงของบิตในอุปกรณ์ โดยบิตแม็พพวกนี้สามารถที่จะนำไปใช้กับอุปกรณ์ได้หลายชนิด

สำหรับการสร้างบิตแม็พนั้น เราต้องกำหนดขนาดและค่าของบิตในบิตแม็พ โดยเรามีวิธีในการสร้างบิตแม็พดังต่อไปนี้

1) ใช้พวกโปรแกรมวาดรูปสร้างบิตแม็พอิมเมจแล้ว save เป็นไฟล์เอาไว้ จากนั้นจึง load มาใช้ภายหลัง (โดยฟังก์ชัน LoadBitmap)

2) ใช้แอพพลิเคชันสร้างบิตแม็พเปล่า ๆ ขึ้นมา (blank bitmap) แล้วใช้ GDI output functions สร้างภาพอีกที

3) ใช้การลงโค้ดของบิตแม็พในตัวโปรแกรม เช่น char * bmp [ ] = {0×70, 0×7F, 0×15, 0×54};

4) สร้างบิตแม็พแล้วกำหนดค่าเริ่มต้นของภาพ (initial image) โดยใช้ฟังก์ชันชื่อ SeDIBits To Device ภาพที่เกิดจาก device-independent bitmap

ในบรรดาพวกเราที่คลุกคลีอยู่กับคอมพิวเตอร์ไม่ว่ามากหรือน้อย มักจะเคยได้ยินคำว่าบิตแม็พนี้อยู่เสมอ โดยเมื่อพูดถึงก็จะพากันนึกถึงรูปภาพที่แสดงบนจอภาพของโปรแกรมกราฟิก

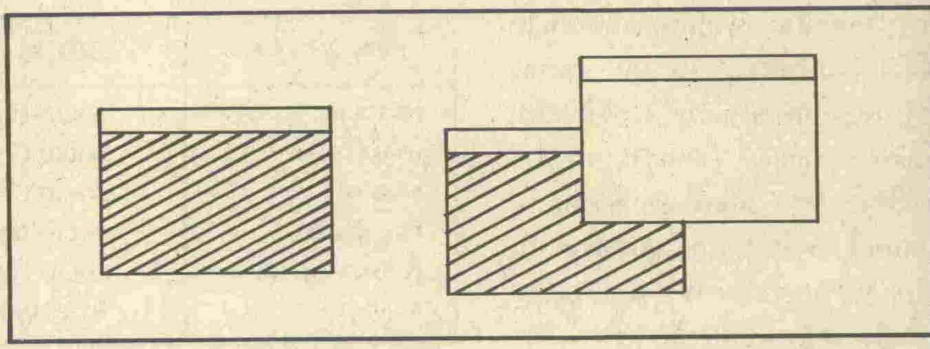
(ซึ่งรวมถึง GUIs ด้วย) โดยเก็บข้อมูลของภาพนั้นไว้ในหน่วยความจำ ซึ่งมันก็มีประโยชน์อยู่ตรงที่มันสามารถเคลื่อนย้ายข้อมูลจากหน่วยความจำไปบนจอภาพได้อย่างรวดเร็ว ซึ่งเร็วกว่าการสร้างรูปภาพนั้นด้วยฟังก์ชันกราฟิก แต่มันก็มีข้อเสียอยู่นะครับ

ประการแรก บิตแม็พจะเป็นข้อมูลล้วน ๆ ที่ประกอบกันเป็นรูป มันไม่มีข้อมูลเกี่ยวกับกรรมวิธีการสร้างรูปนั้นอยู่เลย ดังนั้นการแก้ไขรูปที่เกิดจากบิตแม็พมักทำได้ลำบากกว่ารูปที่สร้างจากการใช้ฟังก์ชันกราฟิกทั้งหลาย สิ่งที่เราสามารถทำได้ในการแก้ไขบิตแม็พก็คือการตัดสินใจว่าจะปิดเปิดบิตไหนเท่านั้น

ประการที่สอง คือมันค่อนข้างจะมีขนาดใหญ่โตทำให้เสียเนื้อที่บนหน่วยความจำไปมาก ยิ่งจอภาพที่มีความละเอียดสูงก็จะยิ่งเสียเนื้อที่มากขึ้นไปอีก อย่างเช่นจอสีจะกินเนื้อที่บนหน่วยความจำมากกว่าบนจอโมโนโครมในขนาดของบิตแม็พที่เท่ากัน ทั้งนี้เพราะบนจอสีต้องมีข้อมูลเกี่ยวกับสีเก็บไว้ในบิตแม็พด้วย

ควบคุมการใช้สอยบนจอภาพ

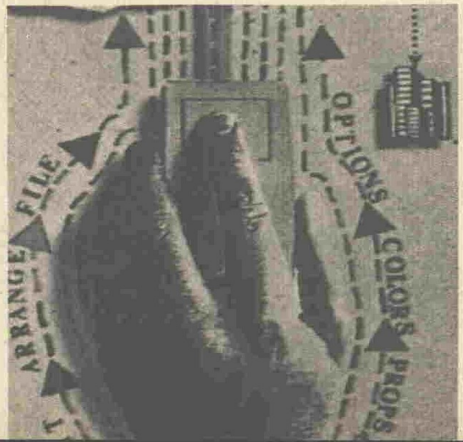
เป็นหน้าที่ของ GDI ในการที่จะไม่ให้บรรดา output functions ทั้งหลายแสดงข้อมูลเกินออกนอกกรอบที่กำลังทำงานอยู่ หรือจะพูดให้ลึก ๆ ก็คือ GDI จะต้องป้องกันไม่ให้มีการใช้พื้นที่เกินขอบเขตของไคลเอนต์แอเรียไปโดนนอน-ไคลเอนต์แอเรียได้ แนวความคิดในการป้องกันบางส่วนของจอภาพจากการวาดโดยเอาต์พุตฟังก์ชันนี้เรียกว่า Clipping บรรดา GDI output functions



รูปที่ 1 แสดงคลิปปิงแอเรียที่ขึ้นกับขนาดของไคลเอนต์แอเรีย

ต่าง ๆ จะถูกกำหนดให้วาดอยู่แต่ในคลิปปิงแอเรีย (ซึ่งอาจไม่ใช่ไคลเอนต์-แอเรียทั้งหมดก็ได้) ส่วนค่าตำแหน่งของคลิปปิงแอเรียนี้ได้ถูกเก็บไว้ในดิสเพลย์-คอนเท็กซ์ซึ่งตามค่า default แล้ว เมื่อมีการสร้างดิสเพลย์คอนเท็กซ์ คลิปปิง-แอเรียจะถูกเซตให้เป็นส่วนของไคล-เอนต์แอเรียที่สามารถมองเห็นได้ในขณะนั้น ดังรูปที่ 1 แต่ในโปรแกรมเราสามารถใช้ GDI functions เพื่อเซตขนาดของคลิปปิงแอเรียได้ แต่โดยทั่วไปต้องมีขนาดไม่เกินกว่าส่วนของไคล-เอนต์แอเรียที่ปรากฏให้เห็นในขณะนั้น ยกเว้นกรณีที่เรากำหนด clipping area เป็น entire window

ถึงแม้ว่าบรรดาแอพพลิเคชันจะไม่ต้องคอยพะวงเกี่ยวกับขนาดของวินโดว์เพราะจะไม่มีเหตุการณ์ที่มีการวาดแล้วทะลักออกมานอกคลิปปิงแอเรีย แต่ส่วนที่แอพพลิเคชันสนใจตอนเปลี่ยนขนาดวินโดว์ ก็คือจะตอบสนองไป



อย่างไร ซึ่งแอพพลิเคชันแต่ละตัวอา
มีการตอบสนองไม่เหมือนกันก็ได้ สิ่
เหล่านี้นำมาสู่การแม็พปิงในขั้นต่อไป

ว่าด้วยการแม็พปิง

GDI functions จะรับค่าพารามิ
เตอร์จากแอพพลิเคชัน ได้แก่ ขนา
ตำแหน่งและความยาวในรูปของลอจิก
คัลโคออร์ดิเนต (logical coordinate
ซึ่งโดยทั่วไปแล้วจะเป็นค่าของ (x,y) ที่น
ไปเปรียบเทียบกับค่าของไคลเอนต์แอเรี
หลังจากนั้นระบบวินโดว์จะแม็พค
เหล่านี้ไปลงบนโคออร์ดิเนตจริงใ
จอภาพ ความสามารถในการทำแม็พปิ
เป็นเรื่องสำคัญมาก ทั้งนี้เพราะระบ
วินโดว์จะต้องยอมให้บรรดาแอพพลิเคชั
ต่าง ๆ ใช้รูทีนเดิมในการวาดเมื่อมีก
refresh window เกิดขึ้น โดยไม่ต้อ
คำนึงถึงขนาดของวินโดว์ในตอนนั้
เช่น เมื่อแอพพลิเคชันจะวาดกราฟแท
การวาดกราฟแท่งจะขึ้นกับค่าที่เป็
อินพุตของรูทีนการวาด เช่น บาร์ โค
ไม่มีการเปลี่ยนแปลงค่าไม่ว่าขนาดข
วินโดว์จะอ้วนเตี้ย ผอมสูงอย่างไร แ
รูปร่างของบาร์จะเปลี่ยนไปให้เหมาะกั
ขนาดของวินโดว์ กล่าวคือ มันไม่สน
ต่อขนาดของวินโดว์หากแต่ไปสนใจ
การตอบสนองต่อการเปลี่ยนแปลงขนา

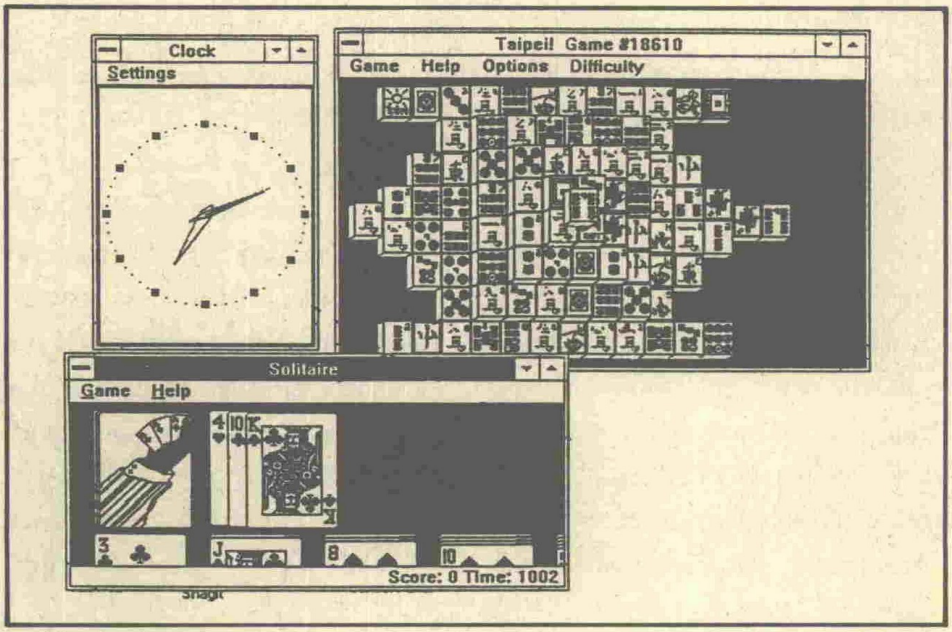
ของวินโดว์แทน ตามปกติแล้วระบบวินโดว์จะไม่รู้เลยว่าแอพพลิเคชันจะมีการตอบสนองต่อการเปลี่ยนแปลงขนาดของวินโดว์อย่างไรบ้าง ส่วนเราผู้เปลี่ยนแปลงขนาดของวินโดว์บนจอภาพ ก็คือผู้จัดสรรการใช้ทรัพยากรบนจอภาพให้กับแต่ละวินโดว์ หรือ (screen utilization controller!!!) นั่นเอง

จากตัวอย่างในรูปที่ 2 และ 3 เราจะเห็นการตอบสนองต่อการเปลี่ยนแปลงขนาดของวินโดว์ที่แตกต่างกันโดย

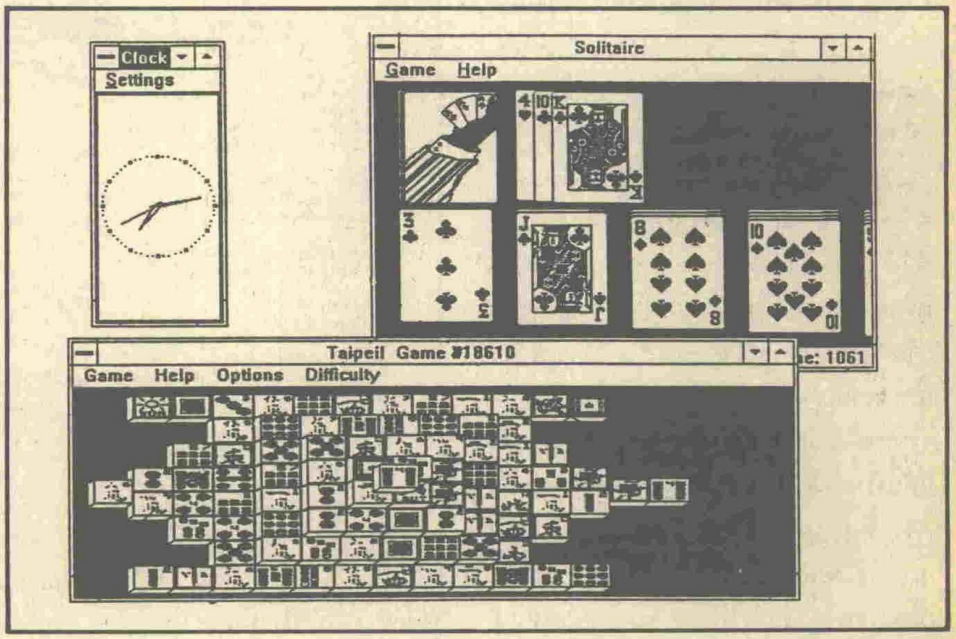
- Application:Solitaire จะรักษาขนาดและรูปร่าง (size and shape) ไว้ตลอดเวลา โดยยอมรับการย่อขนาดที่จะทำให้ข้อมูลบางส่วนหายไป

- Application:Clock ตอบสนองต่อการเปลี่ยนแปลงขนาดวินโดว์โดยการเปลี่ยนแปลงขนาดใหม่ แต่ยังคงรักษาคุณสมบัติส่วนตัวคือรูปร่างให้เป็นวงกลมเสมอ

- Application:Taipei จะเปลี่ยนแปลงทั้งขนาดและรูปร่าง ขอเพียงแต่สามารถเข้ากับขนาดใหม่ของวินโดว์ให้ได้มากที่สุดเท่าที่จะทำได้



รูปที่ 2 แสดงรูปของ Clock, Taipei, Solitaire



รูปที่ 3 แสดงรูปของ Clock, Taipei, Solitaire

จากตัวอย่าง เป็นการแสดงรูปแบบทั่ว ๆ ไปของแอพพลิเคชันจะตอบสนองต่อการเปลี่ยนขนาดของวินโดว์โดย

1) รักษาทั้ง scale และ aspect ratio (Solitaire)

2) รักษา aspect ratio โดยยอมเปลี่ยน scale (Clock)

3) ไม่รักษาทั้ง scale และ aspect ratio (Taipei)

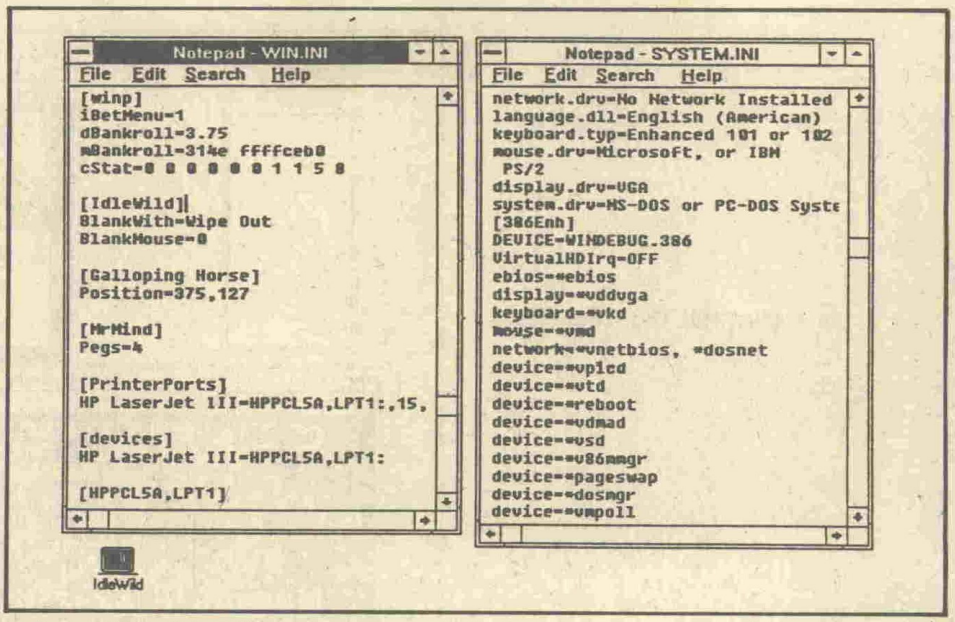
ใน Windows มีแม็พปิงโหมดอยู่ 8 โหมด ดังได้แสดงไว้ในตารางที่ 2

MAPPING MODE	INCREASING VALUES	
	x-axis	y-axis
MM-TEXT	Right	Down
MM-LOMETRIC	Right	Up
MM-HIMETRIC	Right	Up
MM-LOENGLISH	Right	Up
MM-HIENGLISH	Right	Up
MM-TWIPS	Right	Up
MM-ISOTROPIC	Selectable	Selectable
MM-ANISOTROPIC	Selectable	Selectable

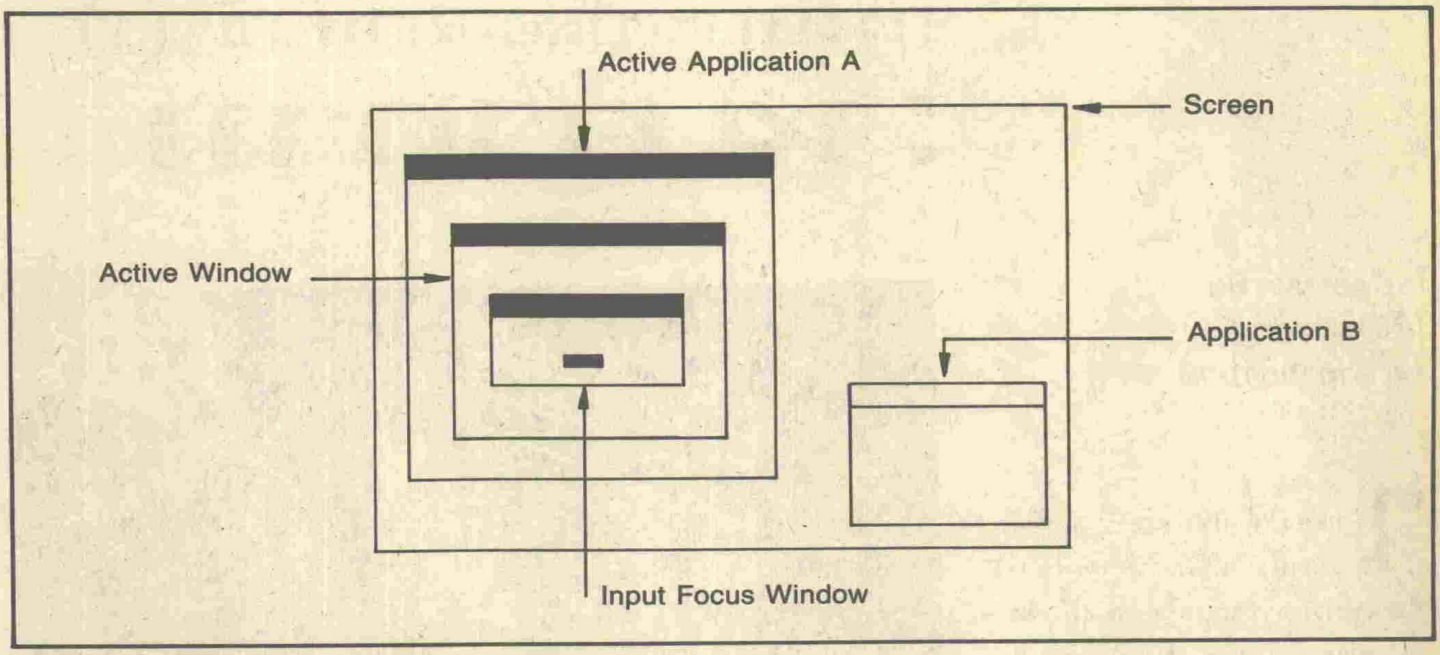
ตารางที่ 2 แสดงข้อมูลบางส่วนของแม็พปิงโหมดทั้ง 8 บน GUIs ที่ชื่อ WINDOWS

สภาพการทำงานของ GUIs บน Multitasking

คุณสมบัติอีกประการหนึ่งที่มักพบใน GUIs ทั่ว ๆ ไปนอกเหนือจากคุณสมบัติทางด้านกราฟิกโอเรียนเต็ด (graphic oriented) ดังที่กล่าวมาตอนต้นคือคุณสมบัติในการทำงานภายใต้ระบบมัลติทาสกิง อันเป็นผลสืบเนื่องมาจากความสามารถของตัว OS (ยกเว้น GUIs บางตัวที่เข้าไปแทรกแซงระบบเอง) ในสภาพการทำงานบนระบบมัลติทาสกิงนั้น แอพพลิเคชันหลายตัวหรือหลาย ๆ ก็อปปี้ของแอพพลิเคชันจะรันไปพร้อม ๆ กัน เราเรียกแต่ละตัวที่วิ่งว่า ทาสก์ (task) สมมติว่าอาจมีผู้ใช้คนหนึ่งรันโปรแกรม NotePad เพื่อตรวจสอบไฟล์ WIN.INI ในขณะเดียวกันเขาก็จะทำการรันโปรแกรม IdleWild เพื่อทำการเซตค่าบางอย่าง จะเห็นได้ว่าในตอนนี้เรามี 2 ทาสก์แล้วที่วิ่งอยู่ ถ้าหลังจากนั้นเขาต้องการดูไฟล์ SYSTEM.INI โดยการรันโปรแกรม NotePad เปิดไฟล์ขึ้นมา และเมื่อเราทำการปิดโปรแกรม IdleWild ไปเป็นไอคอน เก็บไว้ในบริเวณไอคอนแอเรีย ดังแสดงในรูปที่ 4 ซึ่งเมื่อถึงตอนนี้เราพอสรุปได้ว่า มี 3 ทาสก์กำลังรันอยู่ คือ 1 อินสแตนท์ของ IdleWild และ 2 อินสแตนท์ ของ NotePad โดยอินสแตนท์ของแอพพลิเคชันเดียวกันจะแชร์ (share) โค้ดชุดเดียวกันในการทำงานแต่จะแยกข้อมูลกันคนละชุด ถึงแม้ว่าในขณะนี้ จะมีทาสก์วิ่งอยู่ถึง 3 ทาสก์ แต่ก็จะมีเพียงหนึ่งทาสก์เท่านั้นที่จะใช้ตัวโปรเซสเซอร์อยู่ในเวลานั้น ทั้งนี้เป็นเพราะบรรดาคอมพิวเตอร์พีซีทั้งหลายล้วนแล้วแต่เป็นเครื่องในลักษณะ sequential ไม่ใช่ concurrent processors จึงทำให้มันสามารถรองรับการทำงานได้ทีละหนึ่งเท่านั้น ความแตกต่างระหว่างงานที่กำลังโปรเซสกับงานที่กำลังรันอยู่มี

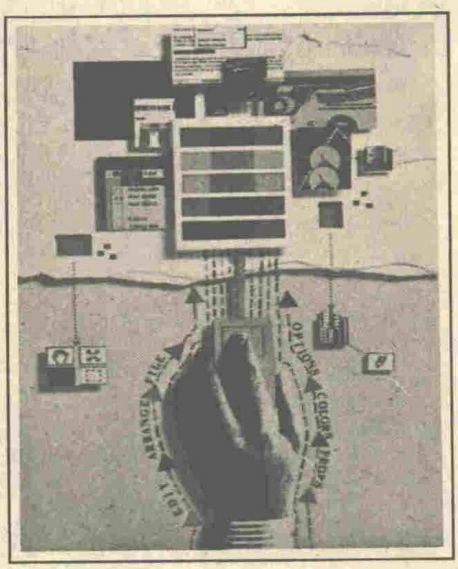


รูปที่ 4 แสดงรูปของ Notepad



รูปที่ 5 แสดงแอคตีฟแอพพลิเคชัน แอคตีฟวินโดว์ และ อินพุตโฟกัสวินโดว์

ความหมายแตกต่างกันมาก (ในที่นี้ใช้ความหมายของ 1 ทาสก์ เทียบเท่ากับ 1 โปรเซส ถึงจุดนี้ก็เป็นอันเข้าใจกันว่าจะมีเพียง 1 แอพพลิเคชัน เท่านั้นที่จะได้รับการโปรเซสในช่วงเวลาหนึ่ง แล้วหมุนเวียนการใช้โปรเซสเซอร์ไปให้ทาสก์อื่น ๆ ซึ่งอาจเป็นแอพพลิเคชันคนละตัวกันหรือแอพพลิเคชันตัวเดียวกันแต่คนละอินสแตนท์ก็ได้ แต่ถ้าเราสังเกตจะเห็นได้ว่ามีเพียง 1 วินโดว์ของ 1 อินสแตนท์ในแอพพลิเคชันตัวหนึ่งเท่านั้นที่จะได้รับค่าทางคีย์บอร์ด ซึ่งวินโดว์นั้นอาจเป็นไชล์ดวินโดว์ (child window) ก็ได้ โดยเราจะเรียกวินโดว์นี้อย่างเป็นทางการว่าอินพุตโฟกัสวินโดว์ (input focus window) โดยอินพุตโฟกัสวินโดว์จะเกิดภายใต้แอคตีฟแอพพลิเคชัน (active application) เท่านั้น (สังเกตได้จากไฮไลท์ Titlebar) นอกจากนั้น ถ้าจะพูดกันให้ชัดเจนยิ่งขึ้น ก็คือ ในแอคตีฟแอพพลิเคชันมี 1 แอคตีฟวินโดว์ ซึ่งภายในก็จะมีอินพุตโฟกัสวินโดว์อยู่ด้วย หรืออาจลองนึกถึงแอคตีฟแอพพลิเคชันของแอพพลิเคชัน A ซึ่งมีไชล์ดวินโดว์ Aa เป็นแอคตีฟวินโดว์ทำการเปิดไดอะลอกวินโดว์ เพื่อเป็นอินพุตโฟกัสวินโดว์ อย่างไรก็ตาม ทั้งแอคตีฟแอพพลิเคชัน แอคตีฟวินโดว์ และอินพุตโฟกัสวินโดว์อาจจะเป็นตัวเดียวกันก็ได้ในบางแอพพลิเคชัน



สุดท้ายนี้ในระบบ GUIs ที่ทำงานแบบมัลติทาสกิงจะต้องอาศัยกลไกในการจัดแบ่งจังหวะปฏิบัติงานของโปรเซสเซอร์ให้แก่บรรดาแอพพลิเคชันที่วิ่งอยู่ในขณะนั้น โดยอาศัยเทคนิคของการจัดการเมสเสจ (message) มาช่วย ซึ่งฉบับหน้า เราจะมาพูดถึงบทบาทของเมสเสจที่มีต่อการทำงานในระบบมัลติทาสกิง โครงสร้างของเมสเสจ, ชนิดของเมสเสจ ฯลฯ พบกันใหม่ฉบับหน้าครับ